Forecasting Intermittent Time Series Using Gaussian Processes

MAU44M00 Report

Sebastian Chejniak

March 29, 2022

Abstract

Intermittent time series (i.e. those that come in the form of counts and have many zero counts), are notoriously difficult to forecast, as they violate basic assumptions made by many classical forecasting methods. Despite this, they occur frequently in retail contexts within many industries [1]. As such, intermittent time series forecasting is an important task for many organisations. In this thesis, I discuss, and test the effectiveness of, a new Gaussian process-based forecasting method in application to the large, hierarchical dataset from the M5 competition [2]. This method was inspired on a method developed in part by my supervisor [3], with the major difference being the use of a negative binomial likelihood instead of a Gaussian likelihood. The new method is found to be outperformed by the baseline methods used. However, I argue that the method can naturally be extended to a hierarchical version thereof, which would be able to leverage the hierarchical structure of the M5 dataset to potentially yield better results. Therefore I suggest this as a worthwhile avenue for further research, especially considering this thesis and the corresponding source code [4] has laid much of the groundwork for such research.

Contents

1 Introduction

Time series are sequential data which are indexed with time. They can take the form of stock prices, GDP, electricity demand, etc. Intermittent time series are those which contain data which takes only non-negative integer ("count") values, where there are mostly zero valued counts. Intermittent data often arises where data is progressively disaggregated and where the intervals between times steps are lower [5]. It is omnipresent in industries that rely heavily on after-sales support, such as the automotive, IT, and

electronics industries [1]. It often occurs as sales data in a retail environment, such as in the case of a large online retailer like Amazon. Numerous reports show that intermittent demand forecasting is one of the major issues facing modern organisations [1].

The count nature of intermittent data means that assumptions such as Gaussian distributed noise are violated. It makes the use of many traditional time series forecasting methods such as ARIMA and SES, designed for continuous or "approximately continuous" data, grossly inappropriate. As such, studying and forecasting intermittent data requires one to adopt a different approach than would be typically used in forecasting. Models for forecasting intermittent data take approaches such as explicitly splitting the time series into zero and non-zero components [6], and adjusting the likelihood of probabilistic forecasting methods so that it is consistent with the intermittent nature of the data.

In this thesis, I consider some current forecasting methods in application to intermittent data. I use these methods as baselines to research the effectiveness of a new method (which I refer to as the GP method) that is a modification of a successful automatic forecasting method developed in part by my supervisor in 2020, which utilises Gaussian processes [3]. The major difference is that this method uses a negative binomial likelihood instead of the Gaussian likelihood used in [3]. I apply all of these methods to the dataset from the M5 competition [2], which was held in 2020. I discuss how the hierarchical nature of the M5 dataset, as well as the Bayesian nature of the method in [3] motivates the extension of the GP method to a hierarchical model.

In section 2, I describe the methods which I use as baselines in detail, explaining how they deal with the intermittent nature of the data. In section 3 I describe the theory necessary to understand the GP method, and then explain the method itself. Following this, I discuss the (rather non-trivial) structure of the M5 dataset in 4.1, and give an exposition on the metrics which I used to evaluate performance, as well as the justifications of their use and theoretical properties in 4.2. Then, I present the results of all the methods on the M5 data in section 4.3. In section 5, I discuss the conclusions of my experiments.

2 Baseline methods

2.1 Croston method

Note: Here and in section 2.2, for a time series $x = x_1, \ldots, x_n$ with I denote by $\hat{x}_{k|l}$ the forecast of x_k, given/using the data $x_l, x_{l-1}, \ldots, x_1$. The Croston method was developed in 1972 by J. D. Croston [6]. While it does not fully deal with the count nature of time-series data (it can return non-integer forecasts), it is still useful for intermittent data [7]. It was the first forecasting method specifically targeted at intermittent data. While it has certain flaws that more recent versions of the method have addressed, it still performs better on intermittent data than other classical methods such as ARIMA and SES. Due to this and the simplicity of the method, it serves as a useful baseline for evaluating the performance of new models.

Croston's method addresses intermittency by considering two elements of such time series: the size of non-zero counts, and the time between non-zero counts. Therefore, the method splits up the time series into two new series, one measuring all non-zero counts, and one measuring how many zero counts occur between non-zero counts. Say we have a time series y_t, $t = 1, \ldots, T$ of positive counts, where j of these are non-zero. Let q_i, $i = 1, 2, \ldots, j$ be the ith non-zero count, and let a_i be one plus the number of zero counts between q_i and q_{i-1} (i.e. the time between q_i and q_{i-1}). q_i and a_i are typically called the demand and inter-arrival time, respectively.

Let $\hat{q}_{i+1|i}$ and $\hat{a}_{i+1|i}$ be the one-step forecasts of the $(i+1)$th demand and inter-arrival time respectively. The Croston method applies exponential smoothing to q and a, and then models the demand (for the observed time steps) as the quotient of the forecasted demand and inter-arrival time:

$$\hat{q}_{i+1|i} = (1-\alpha)\hat{q}_{i|i-1} + \alpha q_i \tag{1}$$

$$\hat{a}_{i+1|i} = (1-\alpha)\hat{a}_{i|i-1} + \alpha a_i \tag{2}$$

$$\hat{y}_{t|t-1} = \hat{q}_{i|i-1} / \hat{a}_{i|i-1} \tag{3}$$

Where $\alpha \in [0, 1]$ is an exponential smoothing parameter, and $\hat{y}_{t|t-1}$ is the demand forecast for a time after the time corresponding to a_i and q_i, but before or equal to the time corresponding to a_{i+1} and q_{i+1}. Recalling that a_j and q_j correspond to the last observed non-zero demand and inter-arrival time, the h-step ahead forecast for the unobserved demand at time $T + h$ is

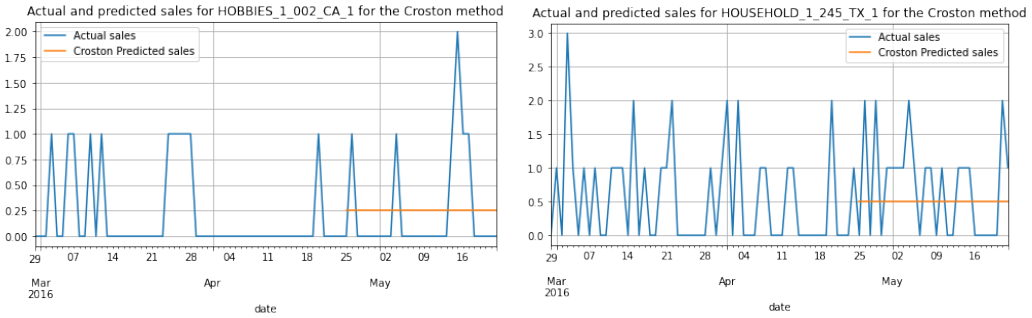

Figure 1: Sales data for the last 74 days of two items in the M5 dataset and forecasted sales for the last 28 days using the Croston Method

then given by

$$\hat{y}_{T+h|T} = \hat{q}_{j+1|j}/\hat{a}_{j+1|j} \tag{4}$$

To choose the parameter α, I used least squares estimation, though this is not inherently a part of the Croston method. Note that the dependence on α in the right hand side of equation (5) is implicit in $\hat{y}_{t|t-1}$

$$\hat{\alpha} = \underset{\alpha}{\operatorname{argmin}} \sum_{t=1}^{T} (\hat{y}_{t|t-1} - y_t)^2 \tag{5}$$

Sample Croston forecasts of two time series in the M5 dataset are given in figure 1

2.2 Croston TSB method

As mentioned in 2.1, Croston is a rather simplistic method, and it has certain flaws. Namely

1. The Croston Method only updates the forecasts after periods of non-zero demand. Therefore, the forecast is not up to date after many periods with zero demand.

2. The Croston Method is positively biased, as was found in [8]

The Croston TSB method [9] addresses these flaws; it is both unbiased and allows for the demand forecasts to decrease during periods of zero demand. The method is similar to the Croston method. The differences between this and the Croston method can be summarised as:

- That the inter-arrival time a_i for $i = 1, \ldots, j$ is replaced with what we call the probability of non-zero demand $p_t \equiv \begin{cases} 1 \text{ if demand occurs at time t} \\ 0 \text{ if demand does not occur at time t} \end{cases}$ for $t = 1, \ldots, T$.
- $\hat{p}_i$, the analog of $\hat{a}_i$ in this method, is updated on all time steps, as opposed to only on non-zero time steps as was done in Croston's method. The demand estimate (which we now call $\hat{z}_t$) is still only updated for periods of non-zero counts.
- Separate exponential smoothing constants α, β are used for the demand and demand probability, while for the Croston method one smoothing constant α was used for both demand and inter-arrival time.

In this method, we do not take note of demand as q_i (which only holds non-zero counts) like we did in the original Croston method. That is, we do not create new time series whose elements correspond only to periods of non-zero demand. All we use in our training data are the time series' p and y.

Note that $\hat{z}_t$ is our estimate of the last non-zero demand before time t, which we find by applying exponential smoothing to y (but only for non-zero y_t). This contrasts with $\hat{y}_t$, which is our estimate of the demand at time t. The key distinction is that $\hat{y}_t$ takes into account the estimated probability of non-zero demand, while $\hat{z}_t$ does not. The method is therefore summarised in the following equations:

$$p_t = 0 \implies \begin{cases} \hat{p}_t = \beta p_t + (1-\beta)\hat{p}_{t-1} = (1-\beta)\hat{p}_{t-1} \\ \hat{z}_t = \hat{z}_{t-1} \\ \hat{y}_t = \hat{p}_t \hat{z}_t \end{cases} \tag{6}$$

$$p_t = 1 \implies \begin{cases} \hat{p}_t = \beta p_t + (1-\beta)\hat{p}_{t-1} \\ \hat{z}_t = \alpha y_t + (1-\alpha)\hat{z}_{t-1} \\ \hat{y}_t = \hat{p}_t \hat{z}_t \end{cases} \tag{7}$$

The h-step ahead forecast for unobserved demand at time $T+h$ is then given by

$$\hat{y}_{T+h|T} = \hat{p}_T \hat{a}_T \tag{8}$$

As in the Croston method, I estimate the parameters (α and β) by least squares estimation:

$$(\hat{\alpha}, \hat{\beta}) = \underset{(\alpha,\beta)}{\operatorname{argmin}} \sum_{t=1}^{T} (\hat{y}_t - y_t)^2 \tag{9}$$

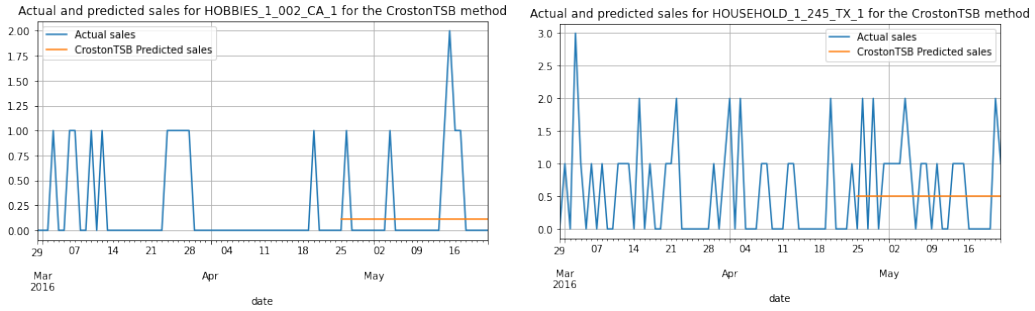

Figure 2: Sales data for the last 74 days of two items in the M5 dataset and forecasted sales for the last 28 days using the CrostonTSB Method

Sample Croston TSB forecasts of two time series in the M5 dataset are given in figure 2

2.3 DeepAR

DeepAR is a deep learning-based probabilistic forecasting method. It has shown great performance when used to forecast a large number of related time-series [10]. This is consistent with the empirical observation that deep learning is excellent at automatically extracting meaningful features from large amounts of high-dimensional data [11] [12]. DeepAR, being a deep-learning method, is basically a "black box", and so a high-level understanding of its structure is sufficient for our purposes. What follows is a high-level explanation of the overall mechanics of deepAR, as well as empirical justifications for the artificial neural network architecture it is based on. All of the following explanation is based on [10]

DeepAR is based on an encoder-decoder recurrent neural network (RNN) architecture with long-short-term-memory(LSTM) cells, which was developed for use in and has seen major success in natural language processing [13]. It has been successfully used for other tasks, such as image processing [14].

An encoder-decoder RNN essentially involves two individual RNNs (the encoder and decoder RNNs respectively). Specific details of the architecture of these RNNs, along with graphics to ease understanding thereof can be found in [10].

The encoder RNN is fed the input, and then the output (typically the

activation at the end of the encoder) of this encoder RNN is fed into the decoder, which outputs the prediction of the desired sequence. The encoder-decoder framework of a neural network allows the input and output sequences to be different lengths, while using a single RNN would require the lengths to be the same.

Recurrent neural networks using LSTM cells have been found to alleviate the issues of vanishing/exploding gradients that plagued traditional RNNs. These are issues which lead to weights near the beginning of the RNN not having much of an effect/having an unreasonably strong effect (respectively) on the gradient of the loss function, therefore causing the model to be trained poorly.

The encoder-decoder aspect of deepAR in the context of time series forecasting means that the model can take as input a time series of a given length, and output a time series of a different length. For example, deepAR can use 56 preceding time steps to forecast the following 28 time steps of a time series. Moreover, the architecture of deepAR allows one to include as input not only the values of the time series', but also a covariate vector for each time step. Such covariate time series' can include other information deemed relevant for prediction of the main time series. For example, it can include the item price on a given day, or other identifying item details. It is through such covariate time series' that deepAR can be trained on and applied to many related time series' in a global manner.

My implementation of deepAR uses: increasing and standardised age, day-of-week, month-of-year, and year covariates, as well as categorical covariates of the item department, category, store, and state. The M5 dataset also provides data regarding special events (e.g. holidays) on each day, as well as item prices. However price data was not included for all time steps, and the PC (8GB RAM) I used to run the code did not have enough RAM to create covariates out of the event data. Given this, and time constraints, I chose to omit these covariates. Including them in future comparisons may be worthwhile, but the method still performs quite well without these covariates, and deepAR is not the main focus of this project.

2.3.1 The model

DeepAR aims to model, for each time series, the probability distribution of the time series in the prediction[1] range, given the values of the time series in the conditioning range, as well as certain covariates for both ranges. This is the distribution $P(\mathbf{y}_{i,t_0:T}|\mathbf{y}_{i,1:t_0-1}, \mathbf{x}_{i,1:T})$, where I use the notation $\mathbf{v}_{i,a:b} = \{v_{i,a}, v_{i,a+1}, \ldots, v_{i,b}\}$ and i is the index labelling the given time series. $y_{i,t} \in \mathbb{Z}_{\geq 0}$ and $x_{i,t} \in \mathbb{R}^k$ denote the value of the ith time series and the vector of covariates for the ith time series at time t, respectively. $[1, t_0 - 1]$ and $[t_0, T]$ are the conditioning and prediction ranges (corresponding to the encoder and decoder networks, respectively), where the former are the indices of y_i for conditioning and the latter are the indices of y_i for prediction. At step t, the inputs of the model are $y_{i,t-1}$, $x_{i,t}$, $\mathbf{h}_{i,t-1}$ (the demand at the previous time step (t-1), the covariate vector at the current time step t, and the output of the preceding sequence of LSTM cells). The output at step t is then $\mathbf{h}_{i,t} = h(\mathbf{h}_{i,t-1}, y_{i,t-1}, x_{i,t}, \Theta)$, where h is the function whose output is that of the sequence of LSTM cells with parameters (weights) Θ. This output is then used to find the parameters of the likelihood $\theta_{i,t} = \theta(\mathbf{h}_{i,t}, \Theta)$, which models the distribution of y_t. Since deepAR is a probabilistic forecasting method, it outputs a distribution of predictions. It does this in the decoder network as follows:

Set $\widetilde{\mathbf{h}}_{i,t_0-1} \equiv \mathbf{h}_{i,t_0-1}$ and $\tilde{y}_{i,t_0-1} \equiv y_{i,t_0-1}$. For $t = t_0, \ldots, T$: The output of the sequence of LSTMs is $\widetilde{\mathbf{h}}_{i,t} = h(\widetilde{\mathbf{h}}_{i,t-1}, \tilde{y}_{i,t-1}, x_{i,t})$. Sample from $\tilde{y}_{i,t} \sim p(y|\theta(\widetilde{\mathbf{h}}_{i,t}, \Theta))$. This generates one sample trace $\tilde{y}_{i,t_0:T}$.

The above process is repeated multiple times in order to obtain multiple traces, and therefore to generate the desired joint prediction distribution. Note that one cannot simply consider the joint prediction distribution as being an independent and identically distributed combination of the likelihood functions at each time step, each with its own respective estimated parameters. This is because each successive $\widetilde{y}_t$ in the prediction range uses not the ground-truth value of y_{t-1} of the previous cell, but rather a prediction thereof, $\widetilde{y}_{t-1}$. Moreover, $\widetilde{y}_{t-1}$ itself depends on the likelihood function with parameters $\theta_{i,t}$, since it is a sample from this likelihood. This process therefore does not generate independent samples of the likelihood distribution.

Instead, the above process suggests that the model assumes that the

[1]This is not necessarily the test data, and the conditioning range is not necessarily the training data of the M5 dataset. The reason is that deepAR actually uses a windowing procedure, taking different "windows" of the training data during the training phase

model distribution $Q_\Theta(y_{i,t_0:T}|y_{i,1:t_0-1}, x_{i,1:T})$ of $y_{i,t_0:T}$, consisting of a product of likelihood factors:

$$\begin{aligned} Q_\Theta(y_{i,t_0:T}|y_{i,1:t_0-1}, x_{i,1:T}) &= \prod_{t=t_0}^{T} Q_\Theta(y_{i,t}|y_{i,1:t-1}, x_{i,1:T}) \\ &= \prod_{t=t_0}^{T} l(y_{i,t}|\theta(\mathbf{h}_{i,t}, \Theta)) \end{aligned} \tag{10}$$

Θ, the collective parameters of the RNN $h(\cdot)$ and the likelihood parameter mapping $\theta(\cdot)$ are found by using stochastic gradient descent to minimize the log-likelihood corresponding to the above distribution. Sample forecasts of deepAR, using the negative binomial likelihood, are given in figure 3

2.3.2 Likelihood model

The choice of likelihood function in the above affects the model's distribution of the predictions, and as such it should reflect some of our prior belief about the behaviour of the time series. As discussed in 4.1, our data comes in the form of intermittent sale counts which are small enough that this count nature cannot be ignored. Therefore we should choose a likelihood which only allows non-negative integer values of $y_{i,t}$, and which allows for $y_{i,t}$ itself to have a high probability of being zero, with an appropriate choice of parameters. A Gaussian likelihood, which is the default of deepAR, would be very inappropriate for this data.

Just like for the Gaussian process method, this motivates our choice of a negative binomial distribution as the likelihood function - it considers only non-negative integer data and it allows for the possibility of a high probability being assigned to zero valued observations. Details on the parametrisation of the negative binomial distribution used in deepAR, as well as the functions used to map the output of each cell to the parameters of this distribution, can be found in [10]

3 The GP method

3.1 Gaussian processes

Definition 3.1 (Gaussian Process)**.** A Gaussian process [15] $GP(m, k) = \{X_t\}_{t\in T}$ is a set of real valued random variables, all defined on the same

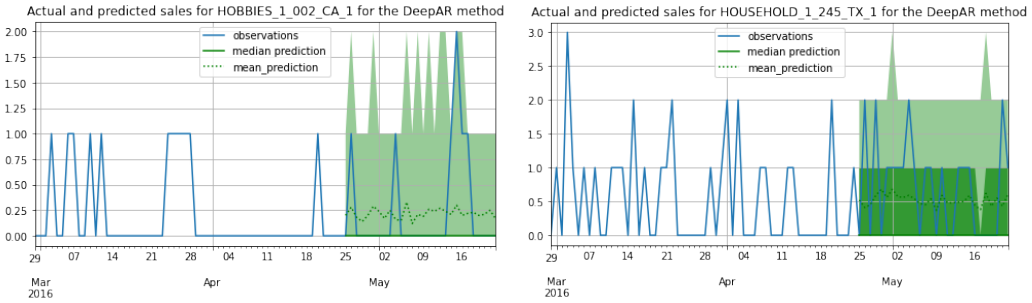

Figure 3: Sales data for the last 74 days of two items in the M5 dataset and forecasted distribution of sales for the last 28 days using the deepAR with the negative binomial distribution as the likelihood. Light green indicates the 90% prediction interval, dark green indicates the 50% prediction interval

probability space, such that for any finite subset $F \subset T$ the random vector $X_F = \{X_t\}_{t \in F}$ has a (multivariate) Gaussian distribution.

Gaussian processes are uniquely defined by their mean and covariance functions. The covariance function k is a symmetric bivariate function which takes as input indices s, t and returns the covariance of the two corresponding Gaussian random variables: $k(s,t) = Cov(X_s, X_t)$. The mean function is a univariate function which takes as input an index s and outputs the mean of the corresponding Gaussian random variable: $m(s) = E[X_s]$ [15]

3.1.1 Gaussian process regression

In a regression context, the Gaussian process is typically thought of as specifying a prior distribution over functions $f : \mathbb{R}^n \mapsto \mathbb{R}$. To realise this view through the above notation, we consider the finite set of random variables from the Gaussian process to correspond to a function f evaluated at some points of $\mathbb{R}^n$: $\{X_i\}_{i=1,\ldots,n} = \{f(\mathbf{x}_i)\}_{i=1,\ldots,n}$. Therefore the index set T is $\mathbb{R}^n$ and the subset F thereof is $\{\mathbf{x}_i\}_{i=1,\ldots,n}$. While the Gaussian process cannot, in general, specify a probability density function over the functions f[2], it does allow us to sample the outputs of functions f on a finite subset of $\mathbb{R}^n$. By definition of $GP(m,k)$ we have that the vector $(f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n))^T$ is

[2]The form of the covariance function decides what functions the Gaussian process considers, and Gaussian processes for some kernel functions can be shown to correspond to functions with an infinite number of parameters.

Normally distributed with mean vector given by $(m(\mathbf{x}_1), m(\mathbf{x}_2), \ldots, m(\mathbf{x}_n))^T$ and $n \times n$ covariance matrix K with i, j-th entry $k(\mathbf{x}_i, \mathbf{x}_j)$. Therefore we can sample from this distribution to sample $(f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n))^T$. One can think of this as sampling from $GP(m, k)$ the subset of functions whose values match that of $(f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n))$. In what follows in section 3, I consider Gaussian processes of the form $GP(0, k)$, i.e. with a mean function $m(\cdot) = 0$. The justification behind this is that by using training data, the Gaussian process can learn the mean function by itself. This is often done in regression and time series forecasting [3]

For regression purposes, consider that, given training data $\{(\mathbf{x}_i, y_i)\}_{i=1,\ldots,n}$, we wish to predict $f(\mathbf{x}_{n+1})$ for some $\mathbf{x}_{n+1}$, assuming $f \sim GP(0, k)$. Then we can find the probability distribution of $f(\mathbf{x})$ by using known properties of conditioning a joint Gaussian distribution as follows:

Let $m = 1$, to make generalising to multiple test points easier through our notation. Note that without observing the training data, the random vector $(f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_{n+m}))^T$ follows[3] a Gaussian Distribution with mean 0 and the $(n + m) \times (n + m)$ covariance matrix with i, j-th entries[4] $k(x_i, x_j)$. So to find a probability distribution of $f(\mathbf{x}_{n+m})$ we can condition the aforementioned Gaussian Distribution on $f(\mathbf{x}_i) = y_i$, $i = 1, \ldots, n$. Let us write the covariance matrix as $\begin{pmatrix} K & K_* \\ K_*^T & K_{**} \end{pmatrix}$, where K, K_*, K_{**} are $n \times n$, $n \times m$, $m \times m$ matrices respectively, and let $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$. Then the conditional probability distribution is [16]:

$$\left(f(\mathbf{x}_{n+m}) \middle| \begin{pmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_n) \end{pmatrix} = \mathbf{y} \right) \sim N(K_* K^{-1} \mathbf{y}, K_{**} - K_*^T K^{-1} K_*) \tag{11}$$

One can easily generalise the above to use the training data to predict a joint distribution of observations at $m > 1$ points $\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \ldots, \mathbf{x}_{n+m}$, by re-

[3] More explicitly, the mean is the n-dimensional zero vector.

[4] This assumes no noise in the observations. Typically, Gaussian independent and identically distributed(iid) noise in the observations is modelled by replacing $k(x_i, x_j)$ with $k(x_i, x_j) + \delta_{ij}$, where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise

placing $f(\mathbf{x}_{n+m})$ in the above with $(f(\mathbf{x}_{n+1}), (\mathbf{x}_{n+2}), \ldots, f(\mathbf{x}_{n+m}))^T$ in places where $f(\mathbf{x}_{n+m})$ is not written inside a vector. That is,

$$\left(\begin{pmatrix} f(\mathbf{x}_{n+1}) \\ f(\mathbf{x}_{n+2}) \\ \vdots \\ f(\mathbf{x}_{n+m}) \end{pmatrix} \middle| \begin{pmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_n) \end{pmatrix} = \mathbf{y} \right) \sim N(K_* K^{-1}\mathbf{y}, K_{**} - K_*^T K^{-1} K_*) \quad (12)$$

Let's introduce some notation for later. I write equation (12), using the more compact notation:

$$p(\mathbf{f}_{test} \mid \mathbf{f}_{train} = \mathbf{y}) \sim N(K_* K^{-1}\mathbf{y}, K_{**} - K_*^T K^{-1} K_*) \quad (13)$$

Denote by $\mathcal{D}$ the collection of the training data $\{(\mathbf{x}_i, y_i)\}_{i=1,\ldots,n}$ and the points at which we wish to perform prediction as $\{\mathbf{x}_i\}_{i=n+1,n+2,\ldots,n+m}$. Then, I denote the joint probability density function of $N(K_* K^{-1}\mathbf{y}, K_{**} - K_*^T K^{-1} K_*)$ by

$$p(\mathbf{f}_{test}|\mathcal{D}) \quad (14)$$

3.1.2 The intuition behind Gaussian processes

As discussed, a Gaussian process is uniquely defined by its mean and covariance functions. The interpretation of the mean function is rather simple – $m(\mathbf{x})$ is the expected value $\mathbb{E}_F[F(\mathbf{x})]$, where the expectation is over the distribution of functions specified by the Gaussian process. Consider the domain to be $\mathbb{R}$ temporarily for illustrative purposes. Given the subset of functions from $GP(0, k)$ which satisfy $f(x) = y$ for some $x, y \in \mathbb{R}$, $k(x, z)$ specifies what values $f(z)$ is most likely to take in the following sense[5]. If $k(x, z)$ is close to zero, then the probability distribution of $f(z)$ over all f is the same prior to conditioning on $f(x) = y$. That is, it follows a univariate normal distribution with mean zero and variance $k(z, z)$. If $k(x, z)$ is not negligible, then conditional on $f(x) = y$ the mean of $f(z)$ (again, over functions f satisfying $f(x) = y$) is scaled: $\mathbb{E}_F[F(z)] = 0 \mapsto \mathbb{E}_F[F(z)] = \frac{k(x,z)}{k(x,x)} f(x)$ and its variance is decreased $Var_F[F(z)] = k(z, z) \mapsto k(z, z) - \frac{k(x,z)^2}{k(x,x)}$. One can therefore see that higher covariances $k(x, z)$ allow one to use an observation

[5]Although of course, we are dealing with probability densities so it is more correct to consider what interval of $\mathbb{R}$ $f(z)$ is most likely to lie in

of $f(x)$ to adjust the prior estimate ($m(z) = 0$) of $f(z)$, and to decrease the uncertainty in said observation[6].

3.2 Gaussian processes for time series forecasting

In time series analysis, a time series is often decomposed into three components: trend, seasonal and cyclic components [17]. Time series models often then model the time series as a sum or product of different time series, each modelling one of these components. In applying Gaussian processes to time series analysis we consider a kernel function that is a sum/product of different kernels, each of which correspond to some sort of relevant behaviour which we expect the time series to exhibit. However, using kernel function compositions that are a sum of kernel functions leads to a posterior distribution that is a sum of Gaussian processes, each corresponding to one of the kernels [18]. As such, using a sum of kernels can be thought of as analogous modelling the time series as a sum of Gaussian processes, each with their corresponding kernel.

A linear trend can be modelled using the linear kernel[7]:

$$\text{LIN: } k_{\boldsymbol{\theta}}(x_1, x_2) = s_b^2 + s_l^2 x_1 x_2 \tag{15}$$

The Gaussian process with the following periodic kernel function considers only periodic functions with a period of p_e

$$\text{PER: } k_{\boldsymbol{\theta}}(x_1, x_2) = s_b^2 \exp\left(-\frac{2\sin^2(\pi |x_1 - x_2| / p_e)}{\ell_p^2}\right) \tag{16}$$

3.3 Automatic forecasting method for continuous data

As mentioned before, the method this thesis proposes is a modification of that proposed in [3]. I summarise the aspects of the method relevant to this thesis: The method uses a kernel function that is a sum of 5 kernel[8] functions:

$$\text{K} = \text{PER} + \text{LIN} + \text{RBF} + \text{SM}_1 + \text{SM}_2 \tag{17}$$

[6]The above results follow trivially by setting $m = n = 1$ in the discussion provided in section 3.1.1

[7]A Gaussian process using this kernel is equivalent to Bayesian Linear Regression, but one should never use a linear kernel by itself, as it is much more computationally efficient to simply use Bayesian Linear Regression[19]

[8]A white noise kernel WN : $k_{\boldsymbol{\theta}}(x_1, x_2) = s_v^2 \delta_{x_1, x_2}$ is also implicitly used for points in the training data

Where the periodic (PER) and linear (LIN) are defined as in section 3.2, RBF is the radial basis function kernel:

$$\text{RBF} : k_{\boldsymbol{\theta}}(x_1, x_2) = s_r^2 \exp\left(-\frac{(x_1 - x_2)^2}{2\ell_r^2}\right) \tag{18}$$

and SM_i is the spectral mixture kernel:

$$\text{SM}_i : k_{\boldsymbol{\theta}}(x_1, x_2) = s_{m_i}^2 \exp\left(-\frac{(x_1 - x_2)^2}{2\ell_{m_i}^2}\right) \cos\left(\frac{x_1 - x_2}{\tau_{m_i}}\right) \tag{19}$$

The method assigns the following prior distributions to the hyperparameters of the kernel:

$$\begin{aligned}
&s_l^2, s_r^2, s_p^2, s_{m_1}^2, s_{m_2}^2, s_v^2 \sim \text{logN}(\nu_s, \lambda_s) \\
&\ell_r \sim \text{logN}(\nu_r, \lambda_\ell) \\
&\ell_p \sim \text{logN}(\nu_p, \lambda_\ell) \\
&\ell_{m_1} \sim \text{logN}(\nu_{m_1}, \lambda_\ell) \\
&\ell_{m_2} \sim \text{logN}(\nu_{m_2}, \lambda_\ell) \\
&\tau_{m_1} \sim \text{logN}(\nu_{t_1}, \lambda_\ell) \\
&\tau_{m_2} \sim \text{logN}(\nu_{t_2}, \lambda_\ell)
\end{aligned} \tag{20}$$

Where a random variable X following a log-normal distribution $\text{logN}(\mu, \sigma^2)$ with mean μ and variance σ^2 has probability density function

$$p_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right) \tag{21}$$

and the parameters of the log-normal distributions are chosen using an empirical Bayes approach, of which details can be found in [3].

3.4 Estimation of the kernel parameters

Equation (13) almost tells us everything needed to write down the probability distribution of the time series modelled by this Gaussian process method, however the model does not choose the kernel parameters a-priori. Instead, it specifies prior distributions over the kernel parameters. Therefore, what remains is to incorporate the prior distributions of the hyperparameters into

equation (13). Denote by Θ the collective parameters of the kernel function we use. Then, make the dependence on these parameters explicit by rewriting the pdf $p(\mathbf{f}_{test}|\mathcal{D})$ (equation (14)) as $p(\mathbf{f}_{test}|\Theta, \mathcal{D})$. What we now call $p(\mathbf{f}_{test}|\mathcal{D})$ is the desired predictive distribution. Of course, here $\mathbf{f}_{train}$ and $\mathbf{f}_{test}$ denote the modelled distributions of the time series at the respective times. Write the prior joint distribution of the kernel parameters as $p(\Theta)$. For example, for the method discussed in [3], these parameters are $s_l^2, s_r^2, s_p^2, s_{m_1}^2, s_{m_2}^2, s_v^2, \ell_r, \ell_p, \ell_{m_1}, \ell_{m_2}, \tau_{m_1}, \tau_{m_2}$ and the joint prior $p(\Theta)$ is simply the product[9] of corresponding pdf's given in equation (20)

Let $p(\mathbf{f}_{train} \mid \Theta)$ be the likelihood of the training data[10]. Following Bayes' Law, the posterior distribution of the parameters is then[11]

$$p(\Theta \mid \mathcal{D}) = \frac{p(\mathbf{f}_{train} \mid \Theta)p(\Theta)}{\int_{\Theta'} p(\mathbf{f}_{train} \mid \Theta')p(\Theta')\,\mathrm{d}\Theta'} \tag{22}$$

and therefore using the multiplicative law of probability, the predictive distribution of the test points of the time series is given by

$$p(\mathbf{f}_{test}|\mathcal{D}) = \int_{\Theta} p(\mathbf{f}_{test}|\Theta, \mathcal{D})p(\Theta|\mathcal{D})\,\mathrm{d}\Theta \tag{23}$$

3.5 The new method

The differences between the method from [3] and our new method are:

- We map each output of the Gaussian process to the mean of a negative binomial distribution. We then estimate the parameter r of the negative binomial using maximum likelihood estimation.
- We use a different, simpler, kernel composition[12]. Namely, we just use the following special case of the Matérn kernel:

$$k(x_1, x_2) = \sigma^2 \frac{\exp\left(-\frac{\sqrt{3}|x_1 - x_2|}{\ell}\right)}{1 + \frac{\sqrt{3}|x_1 - x_2|}{\ell}} \tag{24}$$

[9] Which follows by assuming these priors are independent

[10] So it is given by the pdf of an n-dimensional Multivariate Normal Distribution evaluated at the training data $\mathbf{f}_{train}$, where the mean and covariance matrix are found using the kernel function with parameters Θ, as described in section 3.1

[11] $\mathcal{D}$ contains the dependence on $\mathbf{f}_{train}$

[12] We still denote the parameters of these kernel functions by Θ

- We do not specify priors over the kernel parameters, as preliminary experiments on the training data showed that specifying priors did not lead to much of a performance increase. We set the variance $\sigma^2 = 1$ and the lengthscale $\ell = 0.5$

The justification behind using a simple kernel with no priors, as opposed to a more complicated kernel with priors, is that preliminary experiments on the training data revealed that the method performed more poorly on intermittent data when kernels were composed and priors were included. This seems to be a consequence of a lack of signal in the data - an issue that would hopefully be addressed by using a hierarchical model that is fit on multiple time series at once. The mechanics of this model are as follows: We use the parametrisation of the negative binomial distribution with the following probability mass function

$$p_{NB}(y \mid \mu, r) = \binom{y + r - 1}{r - 1} \left(\frac{r}{\mu + r}\right)^r \left(\frac{\mu}{\mu + r}\right)^y \tag{25}$$

which has support $y = 0, 1, \ldots$. We do not directly observe $\mathbf{f}_{train}$, but instead observe $\mathbf{y}_{train}$, so we consider the conditional distribution $p(\mathbf{y}_{test} \mid \mathbf{y}_{train})$ instead of $p(\mathbf{f}_{test} \mid \mathbf{f}_{train})$. To write down the mass function of this conditional distribution, we first need to define the joint distribution $p(\mathbf{y} \mid \Theta, r)$. We model this by assuming that the data is generated by a series of independent negative binomial distributions, whose means are samples of the Gaussian process prior at the corresponding time steps. As the outputs of the Gaussian process are not observed, we use the partition rule to average these out, which leads to the following joint mass function of the counts:

$$p(\mathbf{y}|\Theta, \mathbf{r}) = \int_{\mathbf{f}} \left(\prod_{i=1}^{n+m} p_{\mathrm{NB}}(y_i \mid \mu = e^{f_i}, r_i) \right) p(\mathbf{f} \mid \Theta) \, \mathrm{d}\mathbf{f} \tag{26}$$

Therefore it follows that the marginal for the training data is:

$$\begin{aligned} p(\mathbf{y}_{train}|\Theta, \mathbf{r}) &= \sum_{\mathbf{y}_{test}} \int_{\mathbf{f}} \left(\prod_{i=1}^{n+m} p_{\mathrm{NB}}(y_i \mid \mu = e^{f_i}, r_i) \right) p(\mathbf{f} \mid \Theta) \, \mathrm{d}\mathbf{f} \\ &= \int_{\mathbf{f}} \left(\prod_{i=1}^{n+m} \sum_{\mathbf{y}_{test}} p_{\mathrm{NB}}(y_i \mid \mu = e^{f_i}, r_i) \right) p(\mathbf{f} \mid \Theta) \, \mathrm{d}\mathbf{f} \\ &= \int_{\mathbf{f}} \left(\prod_{i=1}^{n} p_{\mathrm{NB}}(y_i \mid \mu = e^{f_i}, r_i) \right) p(\mathbf{f} \mid \Theta) \, \mathrm{d}\mathbf{f} \end{aligned} \tag{27}$$

Which leads to the desired test distribution:

$$p(\mathbf{y}_{test}|\mathbf{y}_{train}, \Theta, \mathbf{r}) = \frac{p(\mathbf{y}|\Theta, \mathbf{r})}{p(\mathbf{y}_{train}, \Theta, \mathbf{r})}$$
$$= \frac{\int_{\mathbf{f}} \left(\prod_{i=1}^{n+m} p_{\mathrm{NB}}(y_i \mid \mu = e^{f_i}, r_i) \right) p(\mathbf{f} \mid \Theta) \, \mathrm{d}\mathbf{f}}{\int_{\mathbf{f}} \left(\prod_{i=1}^{n} p_{\mathrm{NB}}(y_i \mid \mu = e^{f_i}, r_i) \right) p(\mathbf{f} \mid \Theta) \, \mathrm{d}\mathbf{f}} \tag{28}$$

We write this as $p(\mathbf{y}_{test} \mid \Theta, \mathbf{r}, \mathcal{D})$. We estimate $\mathbf{r}$ by maximum likelihood estimation:

$$\hat{\mathbf{r}} = \underset{\mathbf{r}}{\operatorname{argmax}} \, p(\mathbf{y}_{test}|\Theta, \mathbf{r}, \mathcal{D}) \tag{29}$$

so our final model for the distribution of test counts $y_{test,1}, y_{test,2}, \ldots, y_{test,m}$, given observations of training counts $y_{train,1}, y_{train,2}, \ldots, y_{train,n}$ is summarised in the equation:

$$p(\mathbf{y}_{test}|\Theta, \mathcal{D}) = p(\mathbf{y}_{test} \mid \Theta, \hat{\mathbf{r}}, \mathcal{D}) \tag{30}$$

3.6 Towards a hierarchical model

To extend this method towards a hierarchical version thereof, we would assign common priors to related time series. For example, we could assume that the kernel parameters Θ_{ijk} for the each of the time series of an item i in department j of store k are independent samples of a prior distribution common to department j in store k $p(\Theta_{ijk} \mid \Theta_{jk})$, where Θ_{jk} are themselves independent samples from a prior common to the store k $p(\Theta_{jk} \mid \Theta_k)$, and finally the Θ_k are also independent samples from a prior common to all stores $p(\Theta_k)$. The prior of the kernel parameters would therefore be:

$$p(\Theta_{ijk}) = \int_{\Theta_k} \int_{\Theta_{jk}} p(\Theta_{ijk} \mid \Theta_{jk}) p(\Theta_{jk} \mid \Theta_k) p(\Theta_k) \, \mathrm{d}\Theta_{jk} \, \mathrm{d}\Theta_k \tag{31}$$

Given data $\mathcal{D}$, these parameters can be estimated for each time series using Bayes' law. For example, the priors associated with each department can be found using

$$p(\Theta_{jk} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \Theta_{jk}) p(\Theta_{jk})}{\int_{\Theta_{jk}} p(\mathcal{D} \mid \Theta_{jk}) p(\Theta_{jk}) \, \mathrm{d}\Theta_{jk}} \tag{32}$$

Where

$$p(\mathcal{D} \mid \Theta_{jk}) = \int_{\Theta_{ijk}} p(\mathcal{D} \mid \Theta_{ijk}) p(\Theta_{ijk} \mid \Theta_{jk}) \, \mathrm{d}\Theta_{ijk} \tag{33}$$

$$p(\Theta_{jk}) = \int_{\Theta_k} p(\Theta_{jk} \mid \Theta_k) p(\Theta_k) \, \mathrm{d}\Theta_k \tag{34}$$

It is through the learning of these common parameters that the model can forecast multiple related time series.

4 Results

A GitHub repository containing the source code used to produce the results in this section can be found in [4]

4.1 The dataset

The dataset on which the above models were tested is actual sales data in the US provided by Walmart for purposes of the M5 competition [2]. It is provided in a hierarchical structure, where each the time series of the sales of each item is given along with a label of the corresponding department, category and store to which it belongs, as well as the US state in which the store resides. The competition also provides time series data for the prices of each item and details of special events on each date. A graphical representation of the data is shown in figure 4. The high level of disaggregation in the data indeed results in lots of intermittency in many of the time series, as seen in a sample time series in figure 1

This hierarchical structure of the data lends itself to application of deepAR, as well as a hierarchical extension of the GP method. The inclusion of many calendar and item data, and the fact that the data has many (30,490) time series further suggests deepAR as a forecasting method (see section 2.3).

For purposes of model performance evaluation, we may wish to aggregate these constituent time series using the aforementioned hierarchical structure. This process is called reconciliation. Whenever we do this in section 4, we consider the aggregations given by table 1. Due to computational constraints[13], we did not evaluate the performance of our models on the entire M5 dataset, but rather on a random subset thereof, with 305 time series.

[13]Namely that the GP method takes a long time to run on my PC

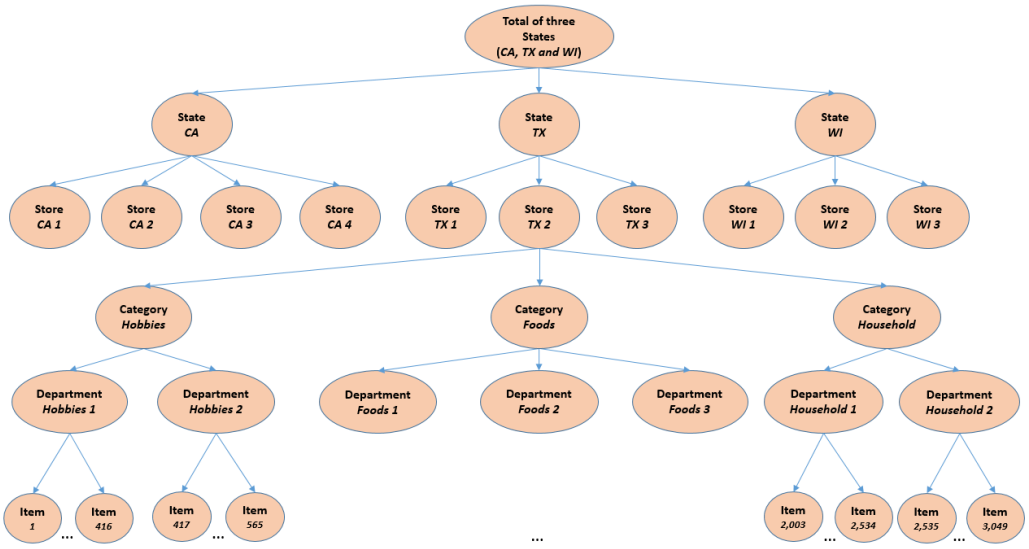

Figure 4: An overview of how the M5 time series are organized [20]

Therefore the number of time series for each level is different, not just in terms of the total amount of time series but also in terms of the proportions of time series per level. I judge the specific details of this difference as unimportant, and therefore I omit explicit details of the structure of the random subset of the M5 dataset this from the report. Therefore, in the discussion of metrics that follows I consider the metrics for the entire M5 dataset, even though the results are given by the metrics on the aforementioned subset of the M5 dataset.

4.2 Metrics

We forecast the time series' on the original data (i.e. forecast the sales for each individual item), and evaluate all of the metrics in section 4.2 on this data, except in the cases of WRMSSE and WSPL. We calculate the WRMSSE and WSPL on the reconciled predictions and data, as described in table 1.

To make this distinction clear, we denote the reconciled data with $\mathcal{Y}$, and the original data with $\mathbf{Y}$. Correspondingly, $\mathcal{Y}_i$ and Y_i are the ith time series' of $\mathcal{Y}$ and $\mathbf{Y}$, respectively (where I leave the fact that $i = 1, \ldots, 42840$ for $\mathcal{Y}$ and $i = 1, \ldots, 30490$ in the case of $\mathbf{Y}$ implicit),

Level id	Aggregation Level	No. of series
1	Unit sales of all products, aggregated for all stores/states	1
2	Unit sales of all products, aggregated for each State	3
3	Unit sales of all products, aggregated for each store	10
4	Unit sales of all products, aggregated for each category	3
5	Unit sales of all products, aggregated for each department	7
6	Unit sales of all products, aggregated for each State and category	9
7	Unit sales of all products, aggregated for each State and department	21
8	Unit sales of all products, aggregated for each store and category	30
9	Unit sales of all products, aggregated for each store and department	70
10	Unit sales of product x, aggregated for all stores/states	3,049
11	Unit sales of product x, aggregated for each State	9,147
12	Unit sales of product x, aggregated for each store	30,490
Total		42,840

Table 1: Number of M5 series per aggregation level [20]

The WRMSSE and the WSPL are the point forecast and the probabilistic forecast performance metrics (respectively) used in the M5 competition [20]. The WRMSSE and WSPL are weighted sums of the RMSSE's and SPL's (respectively) for each time series $\mathcal{Y}_i$ in the reconciled data.

Each of the weights used in this sum, w_i, are calculated as the cumulative dollar sales of the 28 observations of the *test* sample of $\mathcal{Y}_i$, divided by the total (i.e. summed over all test time series' in $\mathcal{Y}$) cumulative dollar sales of the last 28 observations.

Therefore, the WRMSSE and WSPL are metrics which can be thought of as taking into account the relative importance of each time series, using dollar sales as a measure of importance. This makes these metrics particularly useful considering the retail background of the M5 competition. As well as that, this weighting is necessary for the purposes of weighing each level of $\mathcal{Y}$ equally, because some levels (e.g. 1) have substantially less time series than other levels (e.g. 12), but they have many more sales per time series to compensate for this. Due to the structure of this data (more specifically, due to the fact that the sum of sales are equal for all levels), this weighing procedure is equivalent to finding the weighted metrics for each individual level, and then taking the mean of these weighted metrics.

In what follows, let $[1, n]$ be the training interval and let h be the prediction length, so that $[n + 1, n + h]$ is the test interval, for which we wish to forecast sales. In our case, $n = 1913$ and $h = 28$, as we use the first 1913 days of the data for training and forecast the sales for the following 28 days.

4.2.1 Weighted Root Mean Squared Scaled Error (WRMSSE)

Let $\mathcal{Y}_{i,t}$ be the value of the time series $\mathcal{Y}_i$ at time step t, and $\hat{\mathcal{Y}}_{i,t}$ the forecast thereof. Then

$$\text{WRMSSE} = \sum_{i=1}^{42{,}840} w_i \cdot \text{RMSSE}_i \tag{35}$$

Where RMSSE_i is the RMSSE (root mean squared scaled error) of the ith time series:

$$\text{RMSSE}_i = \frac{\sqrt{\frac{1}{h}\sum_{t=n+1}^{n+28}(\mathcal{Y}_{i,t} - \hat{\mathcal{Y}}_{i,t})^2}}{\sqrt{\frac{1}{n-1}\sum_{t=2}^{n}(\mathcal{Y}_{i,t} - \mathcal{Y}_{i,t-1})^2}} \tag{36}$$

That is, the RMSSE is the root mean squared error (RMSE) of the point forecast of $\mathcal{Y}_{i,t}$ in the prediction interval, divided by the RMSE of a forecast of the training data which forecasts the sales on a given day $\mathcal{Y}_t$ as the sales of the previous day $\mathcal{Y}_{t-1}$ (which is called a naive forecast). This rather uncommon choice of metric was justified as follows [20]:

1. The highly intermittent data means that absolute errors assign lower(better) scores to forecasts that forecast the sales as zero. Therefore, to better measure performance, RMSSE uses squared errors, which optimizes for the mean as opposed to the median.

2. This measure is scale-independent, meaning the widely varying scales of $\mathcal{Y}$ are accounted for

3. It can be safely computed without divisions by zero for non-trivial time series'

4. The measure penalizes negative and positive forecast errors equally

4.2.2 Mean Absolute Scaled Error (MASE)

Let $Y_{i,t}$ be the value of the time series Y_i at time step t, and $\hat{Y}_{i,t}$ the forecast thereof. Then we calculate the MASE as

$$\text{MASE} = \frac{1}{30,490}\sum_{i=1}^{30{,}490}\frac{\frac{1}{h}\sum_{t=n+1}^{n+h}|Y_{i,t} - \hat{Y}_{i,t}|}{\frac{1}{n-1}\sum_{t=2}^{n}|Y_{i,t} - Y_{i,t-1}|} \tag{37}$$

Note that this is actually the mean of the MASE for all time series' in $\mathbf{Y}$, as the MASE is usually defined [21]. The MASE is essentially the absolute value version of the RMSSE, analogously to how the mean absolute error(MAE) is the absolute value version of the root-mean squared error. In fact, the RMSSE was actually inspired by the MASE. Therefore, it shares benefits 2-4 from the RMSSE discussed in section 4.2.1. However, (as mentioned in benefit 1) one can argue that it is less suited to intermittent data than the RMSSE due to the fact that it optimizes for the median, and therefore rewards zero forecasts more than the RMSSE.

4.2.3 Weighted Scaled Pinball Loss (WSPL)

The pinball loss function [22] associated with the u-th quantile is given by

$$\begin{aligned} \rho_u(y) &= |y| \begin{cases} u & \text{if } y \leq 0 \\ (1-u) & \text{if } y > 0 \end{cases} \\ &= y(u - \mathbf{I}_{[y \leq 0]}) \end{aligned} \tag{38}$$

Where $\mathbf{I}_{[y \leq 0]}$ is the indicator function that returns 1 if $y \leq 0$ and 0 if $y > 0$. Just like minimising the mean squared error optimises for the mean of a probability distribution, and how minimising the mean absolute error optimises for the median, minimising the pinball loss ρ_u can be shown to optimise for the u-th quantile of a probability distribution [22]. That is, let us be given a probability distribution X, with cumulative distribution function F_X defined over $\mathbb{R}$. Let x_u be the u-th quantile of X, i.e. $x_u = inf\{x \in \mathbb{R} : F_X(x) \geq u\}$. Then $x_u \in \text{argmin}_{x \in \mathbb{R}} \mathbb{E}[\rho_u(X - x)]$. If F_X is strictly increasing, then we have $x_u = \text{argmin}_{x \in \mathbb{R}} \mathbb{E}[\rho_u(X - x)]$

Let $\mathcal{Y}_{i,t}$ be the value of the time series $\mathcal{Y}_i$ at time step t, and $\mathcal{Q}_{i,t}(u)$ the uth quantile forecast thereof. In the M5 competition, the SPL (scaled pinball loss) is given by

$$\text{SPL}_i(u) = \frac{1}{h} \frac{\sum_{t=n+1}^{n+h} \rho_u(\mathcal{Q}_{i,t}(u) - \mathcal{Y}_{i,t})}{\frac{1}{n-1} \sum_{t=2}^{n} |\mathcal{Y}_{i,t} - \mathcal{Y}_{i,t-1}|} \tag{39}$$

That is, it is the mean of the pinball losses for each day in the prediction interval, divided by the mean absolute error of naive forecasts in the training interval.

We calculate the WSPL as a weighted sum of each time series' mean SPL, where the mean is taken over the SPL for each of the 9 quantiles: $u_1 = 0.005$, $u_2 = 0.025$, $u_3 = 0.165$, $u_4 = 0.25$, $u_5 = 0.5$, $u_6 = 0.75$, $u_7 = 0.835$, $u_8 = 0.975$, and $u_9 = 0.995$.

$$\text{WSPL} = \sum_{i=1}^{42{,}840} w_i \cdot \frac{1}{9} \sum_{j=1}^{9} \text{SPL}_i(u_j) \tag{40}$$

Similarly to the RMSSE, benefits [20] of using the SPL include

1. SPL is scale independent
2. It can be safely computed without divisions by zero

4.2.4 Continous Ranked Probability Score (CRPS)

Let F be the empirical cumulative distribution function associated with a probabilistic forecast over $\mathbb{R}$ of an observation x. The CRPS [23] for such a forecast is defined as

$$CRPS(F, x) = \int_{-\infty}^{\infty} (F(y) - \mathbf{I}_{[y \geq x]})^2 dy \tag{41}$$

4.3 Results

As mentioned at the start of this section, we forecasted the time series for a random subset of 305 time series in the M5 dataset. The metrics for each method on this subset are given in table 2. We also found the RMSSE, which is simply the mean RMSSE over each time series in $\mathbf{Y}$. DeepAR, with a WSPL of 0.2, would have scored in the top 20% of competitors in the M5 uncertainty competition, which is also higher than any of the baseline methods used in the M5 competition itself. The GP method, on the other hand, would have scored in the bottom 38% of competitors[14][24]. DeepAR, with a WRMSSE of 0.748, would have scored in the top 16% of the M5 accuracy competition, while the GP method would have scored in the bottom 38%[15][25].

[14] Again, I highlight that I only found forecasts on a subset of the test data, so these results are only approximate, and the score would likely be different if forecasts were performed on the entire M5 dataset.

[15] The fact that this matches its position in the uncertainty competition is a coincidence.

	Dummy	Croston	CrostonTSB	DeepAR	GP
MASE	**1.614**	1.727	1.679	1.695	1.829
RMSSE	1.157	0.943	0.954	**0.915**	0.933
WRMSSE	3.119	0.898	0.901	**0.748**	0.924
CRPS	N/A	N/A	N/A	**0.576**	0.644
WSPL	N/A	N/A	N/A	**0.200**	0.256

Table 2: Performance metrics of each method as well as a Dummy point forecast which forecasted zero sales for all test days. The best performing method for each metric is given in boldface. CRPS and WSPL, being probabilistic metrics, are unavailable for the methods which only provide point forecasts.

5 Conclusions

The accuracy competition, due to the fact that point forecasts are easier to obtain than probabilistic forecasts, had a lower barrier to entry. This was reflected in the higher number of participants (5,558 vs 909). Therefore comparing the percentiles of deepAR's WRMSSE and WSPL between the competitions may be unsuitable, although it does give a rough idea of performance.

The model with the best MASE is the dummy model, which forecasts zero for all test points. As discussed in section 4, the low MASE of the dummy model can simply be explained by the fact that the median of intermittent data is zero, and absolute errors optimise for the median. This result, together with the fact that the dummy model is outperformed on the other point forecast metrics (which use squared errors), provides an empirical basis for the unsuitability of MASE in intermittent time series. Moreover, the WRMSSE of the dummy model is much higher than the WRMSSE of the other methods. This is a consequence of the fact that the WRMSSE considers not only the individual time series, but aggregations thereof. Since the dummy model forecasts zero everywhere, then the higher level aggregations (e.g. the sales summed over all stores) will be forecasted as zero, despite them being much higher than zero in the test data. This illustrates the use of the WRMSSE metric - it is able to punish forecasts which, although seemingly accurate, are not very useful in practice because they just forecast all

sales as zero or almost zero.

The fact that we only used 1% of the time series in the M5 data should have no effect on the performance of the Croston and GP methods, as they only use the counts of a single time series for each forecast. DeepAR, however, utilises the hierarchical structure of the data, and therefore we expect it to perform more poorly on the subset of the M5 dataset as opposed to the entire dataset. That being said, deepAR still outperforms every other method in all metrics but the MASE. The results for the GP method are mediocre at best, as is seen by the fact that its RMSSE is between that of deepAR and Croston's method. The probabilistic forecasts of deepAR are also more accurate than those of the GP method, as seen by the CRPS and WSPL. This suggests that the GP method by itself is unsuitable for forecasting intermittent time series, performing marginally better or more poorly than the much simpler Croston methods.

However, as alluded to before, the GP method can be extended to a hierarchical model. Given the highly hierarchical structure of the M5 dataset, such a model may be able to leverage this structure to provide more accurate forecasts, and it may also benefit from using a larger quantity of data for producing forecasts. Therefore I suggest this as an area of further research, especially as this thesis and the accompanying source code [4] has laid much of the groundwork for such research.

Acknowledgements

This project has been a great learning experience for me both in research and in developing large amounts of code for statistical modelling purposes. I have no doubt that I will carry the lessons I have learned throughout this project with me to industry as well as academia. I thank my supervisor, Dr. Alessio Benavoli, for the opportunity to work on this project and for his constant guidance and feedback throughout it.

References

[1] J. Boylan and A. Syntetos, *Intermittent Demand Forecasting: Context, methods and applications*. 07 2021.

[2] B. Seaman and J. Bowman, "Applicability of the m5 to forecasting at walmart," *International Journal of Forecasting*, 2021.

[3] G. Corani, A. Benavoli, J. Augusto, and M. Zaffalon, "Time series forecasting with gaussian processes needs priors," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases ECML PKDD*, 2021.

[4] https://github.com/SebastianChk/FYP-TCD-Sebastian-Chejniak.

[5] M. Leonard, B. Elsheimer, M. John, and U. Sglavo, "Small improvements causing substantial savings-forecasting intermittent demand data using sas® forecast server," 01 2008.

[6] J. D. Croston, "Forecasting and stock control for intermittent demands," *Operational Research Quarterly (1970-1977)*, vol. 23, no. 3, pp. 289–303, 1972.

[7] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. Australia: OTexts, 3rd ed., 2021.

[8] A. Syntetos and J. Boylan, "On the bias of intermittent demand estimates," *International Journal of Production Economics*, vol. 71, no. 1, pp. 457–466, 2001. Tenth International Symposium on Inventories.

[9] R. H. Teunter, A. A. Syntetos, and M. Zied Babai, "Intermittent demand: Linking forecasting to inventory obsolescence," *European Journal of Operational Research*, vol. 214, no. 3, pp. 606–615, 2011.

[10] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[11] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[12] H. Liang, X. Sun, S. Yunlei, and Y. Gao, "Text feature extraction based on deep learning: a review," *EURASIP Journal on Wireless Communications and Networking*, vol. 2017, 12 2017.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[14] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[15] "University of chicago, statistics 386 course notes." `https://web.archive.org/web/20210506123351/https://galton.uchicago.edu/~lalley/Courses/386/GaussianProcesses.pdf`. Accessed: 27-01-2022.

[16] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[17] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2nd ed., 2018.

[18] "The kernel cookbook: Advice on covariance functions." `https://web.archive.org/web/20220203112912/https://www.cs.toronto.edu/~duvenaud/cookbook/`. Accessed: 03-02-2022.

[19] D. Duvenaud, *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[20] "M5 competition guidelines." `http://web.archive.org/web/20220106104601/https://mofc.unic.ac.cy/wp-content/uploads/2020/03/M5-Competitors-Guide-Final-10-March-2020.docx`. Accessed: 06-01-2022.

[21] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[22] G. Biau and B. Patra, "Sequential quantile prediction of time series," *Information Theory, IEEE Transactions on*, vol. 57, pp. 1664 – 1674, 04 2011.

[23] T. Gneiting and A. E. Raftery, "Strictly proper scoring rules, prediction, and estimation," *Journal of the American Statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.

[24] https://www.kaggle.com/c/m5-forecasting-uncertainty/leaderboard.

[25] https://www.kaggle.com/c/m5-forecasting-accuracy/leaderboard.